ECE 250 Project 0 Playlist
Design Document

Anthony Xu, UW User ID:a68xu

Spet 27 2022

Overview of classes:

Classes used:

- Playlist Class
- Song Class

Playlist Class

Description:

The playlist class represents the playlist that the user will create in this program. It is a dynamically allocated array of local objects of Song. It also has imported member variables crucial to the functionalities of the array, the member functions to implement the functionalities.

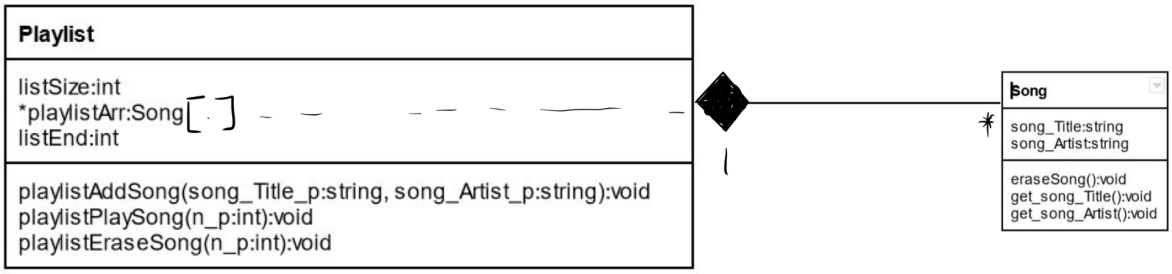
Song Class

Description:

The Song class holds the two attributes of the song: song name and artist. It also takes care of erasing a song(setting it to be an empty song) and housing getters and setters of the song name and artist for outside use

The core part of a Playlist Object is that it has an dynamically allocated array of songs. Where an each individual element of the array(a song object) has the attributes of the song(name and artist)

UML Class diagram



"Composition: In addition to an aggregation relationship, the lifetimes of the objects might be identical, or nearly so." The Playlist Class and the Songs associated with it have same lifecycle if the songs are not erased midway.(in my case the songs objects are never erased, they are just set to empty songs)

From: http://www.cs.utsa.edu/~cs3443/uml/uml.html

Details on design decisions

Playlist Class

Member Variables

```
private:
    int listSize;
    int index_n;
    Song *playlistArr;
    int listEnd;
```

- int listSize stores the size of the array, lets say if the user inputs m 10, then the array will be initiated with a size of 10. listSize is kept track of since we can use it to determine whether we can insert more songs(we can't insert more songs if we already have 10 songs in the list and list end is at the end)
- Song *playlistArr stores the address of the start of the array that we will allocate when initializing the playlist
- Int listEnd stores the index of the array of the end of the list

Member Functions

```
public:
    Playlist(int listSize_p);
    Playlist();
    ~Playlist();
    void playlistAddSong(string song_Title_p, string song_Artist_p);
    void playlistPlaySong(int n_p);
    void playlistEraseSong(int n_p);
```

- Playlist(int listSize_p); this is where the array is dynamically allocated, the size of the array is given when creating the object using the constructor
- Playlist(); this is the default constructor, which will not be tested according to Piazza(probably)
- ~Playlist(); this is the deconstructor, where the array is deallocated and the pointer to the head of the list is set to nullptr
- void playlistAddSong(string song_Title_p, string song_Artist_p); It is appropriate to use the CONST keyword for this function, since we shouldn't be modifying the parameters when adding the songs to the list. CONST would add extra security
- void playlistPlaySong(int n_p); It is appropriate to use the CONST keyword for this function, since we shouldn't be modifying the index of the song we want to play inside the function
- void playlistEraseSong(int n_p); It is appropriate to use the CONST keyword for this function, since we shouldn't be modifying the index of the song we want to erase

Song Class

Member Variables

```
private:
    string song_Title;
    string song_Artist;
```

- int listSize stores the size of the array, lets say if the user inputs m 10, then the array will be initiated with a size of 10. listSize is kept track of since we can use it to determine whether we can insert more songs(we can't insert more songs if we already have 10 songs in the list and list end is at the end)

- Song *playlistArr stores the address of the start of the array that we will allocate when initializing the playlist
- Int listEnd stores the index of the array of the end of the list

Member Functions

```
public:
    Song(string song_Title, string song_Artist);
    Song();
    ~Song();
    void eraseSong();
    string get_song_Title();

    string get_song_Artist();
```

- Song(`string song_Title, string song_Artist`); This constructor is used when we insert a new song to the playlist. When inserting a new song, we create a new Song object with the attributes of the song we want to insert, and assign it to the array at the end of the list index, so we just set the two private member variables with the parameter.
- Song(); this is the default constructor, which will be called when the Playlist class is initiated, so we populate each Song in the array to be an empty Song
- ~Song(); since all Songs are elements inside the array or declared as local variables, I left the deconstructor blank as it will automatically be deconstructed when we deconstruct the playlist
- void eraseSong(); instead of erasing songs by deconstructing the song object, we set it to be an empty song since Piazza mentioned an empty song will not be inserted. If we erase the Song by calling the destructor, we would be deconstructing the object a second time when the playlist is deconstructed. To avoid that, we just set it to be an empty song to indicate that the song is erased from the play list.
  no need to use the CONST keyword when erasing a song

Test Cases

1. Test basic functionalities such as creating the playlist, inserting, playing and erasing. Verify that the output matches the required
2. Test if the edge cases are behaving as normal,(the My Heart Will Go On and Muskrat Love;Captain and Tennille）
3. Inserting the same song many times(there is no mentioning of can not having the same song in the playlist) , playing them, erasing them, and inserting again, playing them to verify it
4. Test if we insert ArraySize times, and insert again, it should say we can no longer insert. Expand this testing by erasing songs and inserting to the max size again, to see if anything is wrong. Add playing commands in between
5. Test different types of string input(songs with numbers, songs with special characters)
6. Detailed testing of erasing a song(when erasing a song at the head(index 0), all following songs should move up by one) since we want to insert songs at the end of the list every time
   If we erase a song at the end of the list, no reindexing is required
   If we erase a song at the head or middle of the list, we need to move the following songs up,
   After erasing, verify the songs are in the correct position by playing them.
7. Testing of different scenarios which are not a part of the core design of the program(wrong input at beginning, exiting the program with invalid command, etc)